ChatGPT

Proiect Bază de Date – Gestionarea Biletelor de Tren

1. Modelul real și utilitatea sa

Sistemul propus gestionează datele pentru vânzarea și rezervarea biletelor de tren operate de companii publice (ex. CFR) și private. Se stochează informații despre **trasee** (cursă între stații), **stații**, **operatori** (companii feroviare), **trenuri**, **tipuri de tren** (Regio, InterRegio, Accelerat etc.), **tarife** și **categorii de tarif** (adult, student, etc.), precum și despre **călători**, **rezervări** și **bilete**. Modelul relațional permite integrarea acestor entități astfel încât aceeași rută să poată fi folosită de mai mulți operatori (rute independente de operator) și să existe tarife variabile pe categorii de călători. Un design bun reduce redundanța și minimizează anomaliile la actualizare [1], facilitând integritatea datelor și interogări complexe (de ex., calcul tarif, rezervări, statistici).

2. Constrângeri funcționale și de integritate

- **Chei primare**: Fiecare tabel are o coloană (sau un set de coloane) definită ca cheie primară (PK), care identifică unic fiecare înregistrare [2]. Cheia primară nu permite valori NULL sau duplicate [2].
- **Chei externe**: Coloanele care leagă tabelele (FK) impun integritate referențială: de exemplu, `Rezervare.IDCalator` referențiază `Calator.IDCalator`. Astfel, nu se pot introduce bilete sau rezervări care referă călători sau trasee inexistente.
- **Constrângeri specifice**: Se definesc constrângeri `CHECK` și valori implicite pe coloane de categorie. De exemplu, în `Operator` coloana `Tip` poate avea doar valori predefinite ('Public', 'Privat'), iar în `Bilet` coloanele `DataEmitere` pot avea valoarea implicită `SYSDATE`.
- **Unicitate și obligatoriu**: Coloane ca codul trenului sau codul traseului pot fi unice (`UNIQUE`), iar câmpurile critice (ex. `NumeStație`, `NumeOperator`) vor fi `NOT NULL` pentru a evita date incomplete.

3. Definirea entităților și cheilor primare

Entitățile propuse sunt următoarele (PR = cheie primară):

- **Operator**: reprezintă compania feroviară. Atribute: *IDOperator* (PR, integer), *NumeOperator* (varchar), *Tip* (varchar, ex. "Public"/"Privat").
- **Traseu**: o rută feroviară de la o stație de plecare la una de sosire. Atribute: *IDTraseu* (PR, integer), *NumeTraseu* (varchar).
- **Stație**: un punct de oprire pe traseu. Atribute: *IDStatie* (PR, integer), *NumeStatie* (varchar), *Oras* (varchar).
- **Statie_Traseu**: tabel asociativ între Stații și Trasee (relație M:N). Atribute: *IDTraseu* (FK), *IDStatie* (FK), *Ordine* (număr pozițional în traseu). Cheia primară poate fi compusă din (*IDTraseu*, *IDStatie*).
- **TipTren**: categorie tren (Regio, InterRegio etc.). Atribute: *IDTipTren* (PR, integer), *Descriere* (varchar).

- **Tren**: vehicul care circulă pe un traseu sub un operator. Atribute: *IDTren* (PR, integer), *NumeTren* (varchar), *Capacitate* (number), *IDTipTren* (FK), *IDOperator* (FK).
- **Calator**: persoană care călătorește. Atribute: *IDCalator* (PR, integer), *Nume* (varchar), *Prenume* (varchar), *DataNasterii* (date), etc.
- **Rezervare**: rezervarea unui set de bilete de către un călător. Atribute: *IDRezervare* (PR, integer), *IDCalator* (FK), *DataRezervare* (date).
- **CategorieTarif**: tip de tarif (ex. "Adult", "Copil", "Student"). Atribute: *IDCategorie* (PR, integer), *Descriere* (varchar), *ProcentReducere* (number).
- **Tarif**: tarif pe traseu pentru o categorie de tarif. Atribute: *IDTarif* (PR, integer), *IDTraseu* (FK), *IDCategorie* (FK), *Pret* (number). Aceasta permite tarife variabile pe traseu și categorie.
- **Bilet**: biletul propriu-zis emis/cumpărat. Atribute: *IDBilet* (PR, integer), *IDRezervare* (FK, opțional), *IDTren* (FK), *IDTraseu* (FK), *IDStatiePlecare* (FK), *IDStatieSosire* (FK), *IDTarif* (FK), *DataEmitere* (date), *PretBilet* (number).

Cheile primare (PR) ale fiecărei entități sunt indicate mai sus; de exemplu *IDOperator* în tabelul Operator asigură unicitatea fiecărui operator [2].

4. Relații și cardinalități

Relațiile dintre entități și cardinalitățile corespunzătoare sunt:

- **Operator–Traseu**: relație M:N (prin tabela *Operator_Traseu*). Un operator poate deservi mai multe trasee și un traseu poate fi folosit de mai mulți operatori. (Ex: operatorul CFR și alte companii private pot circula pe același traseu).
- **Operator–Tren**: 1:N. Un operator deține mai multe trenuri, iar fiecare tren aparține unui singur operator.
- **TipTren–Tren**: 1:N. Un tip de tren se regăsește la multe trenuri, dar fiecare tren are un singur tip (de ex. un tren "Regio").
- **Traseu–Statie**: M:N (prin tabela *Statie_Traseu*). Un traseu traversează mai multe stații, iar o stație poate fi punct de oprire pe mai multe trasee.
- **Calator–Rezervare**: 1:N. Un călător poate avea mai multe rezervări (de ex. rezervări pentru familii), iar fiecare rezervare aparține unui singur călător.
- **Rezervare–Bilet**: 1:N. O rezervare poate genera mai multe bilete (ex. rezervare de grup cu n bilete), iar fiecare bilet aparține unei singure rezervări (sau niciunei rezervări dacă este vândut direct).
- **Tren–Bilet**: 1:N. Un tren poate avea multe bilete emise, dar fiecare bilet este valabil pe un anumit tren.
- **Traseu–Bilet**: 1:N. Un traseu are multe bilete emise, iar fiecare bilet este pe un traseu dat (prin referința la *IDTraseu*).
- **CategorieTarif–Tarif**: 1:N. O categorie de tarif poate avea multiple înregistrări de tarif (pentru trasee diferite), iar fiecare tarif este pentru o singură categorie.
- **Traseu–Tarif**: 1:N. Un traseu are tarife pentru diferite categorii, fiecare tarif fiind asociat unui singur traseu.

Cardinalitățile de tip 1:1, 1:N sau N:M sunt cele standard în modelarea ER [3]. De exemplu, relația Tren–Operator este 1:N (un operator ▶ multe trenuri), iar relația Operator–Traseu este M:N.

5. Atribute, tipuri de date și constrângeri

Vom descrie fiecare tabel cu atributele sale principale, tipurile de date sugerate și constrângerile relevante:

- **Operator**(*IDOperator* INT PRIMARY KEY, *NumeOperator* VARCHAR2(50) NOT NULL, *Tip* VARCHAR2(10) CHECK (Tip IN ('Public','Privat')) DEFAULT 'Public').
- **Traseu**(*IDTraseu* INT PRIMARY KEY, *NumeTraseu* VARCHAR2(50) NOT NULL).
- **Statie**(*IDStatie* INT PRIMARY KEY, *NumeStatie* VARCHAR2(50) NOT NULL, *Oras* VARCHAR2(50) NOT NULL).
- **Statie_Traseu**(*IDTraseu* INT NOT NULL, *IDStatie* INT NOT NULL, *Ordine* NUMBER NOT NULL, PRIMARY KEY(IDTraseu,IDStatie)). (FK către *Traseu.IDTraseu* și *Statie.IDStatie*.)
- **TipTren**(*IDTipTren* INT PRIMARY KEY, *Descriere* VARCHAR2(20) NOT NULL).
- **Tren**(*IDTren* INT PRIMARY KEY, *NumeTren* VARCHAR2(50), *Capacitate* NUMBER, *IDTipTren* INT NOT NULL, *IDOperator* INT NOT NULL). (FK către *TipTren* și *Operator*; de exemplu *Capacitate* > 0.)
- **Calator**(*IDCalator* INT PRIMARY KEY, *Nume* VARCHAR2(50) NOT NULL, *Prenume* VARCHAR2(50) NOT NULL, *DataNasterii* DATE). (Exemple: *DataNasterii* <= SYSDATE.)
- **Rezervare**(*IDRezervare* INT PRIMARY KEY, *IDCalator* INT NOT NULL, *DataRezervare* DATE DEFAULT SYSDATE). (FK către *Calator*.)
- **CategorieTarif**(*IDCategorie* INT PRIMARY KEY, *Descriere* VARCHAR2(20) NOT NULL, *ProcentReducere* NUMBER DEFAULT 0). (Ex.: `ProcentReducere` între 0 și 100 cu `CHECK`.)
- **Tarif**(*IDTarif* INT PRIMARY KEY, *IDTraseu* INT NOT NULL, *IDCategorie* INT NOT NULL, *Pret* NUMBER NOT NULL). (FK către *Traseu* și *CategorieTarif*.)
- **Bilet**(*IDBilet* INT PRIMARY KEY, *IDRezervare* INT, *IDTren* INT NOT NULL, *IDTraseu* INT NOT NULL, *IDStatiePlecare* INT NOT NULL, *IDStatieSosire* INT NOT NULL, *IDTarif* INT NOT NULL, *DataEmitere* DATE DEFAULT SYSDATE, *PretBilet* NUMBER). (FK către *Rezervare* (opțional), *Tren*, *Traseu*, *Statie* (de două ori) și *Tarif*.)

Pentru fiecare tabel s-au stabilit tipuri corespunzătoare (INT pentru chei, VARCHAR2 pentru texte, DATE pentru date calendaristice) și constrângeri (PK, NOT NULL, CHECK, DEFAULT) pentru integritate. De exemplu, **Tip** tren este limitat la valori prestabilite, iar *ID*-urile cheilor străine vor face referință doar la valori existente.

6. Diagrama entitate-relație

Pentru ilustrarea conceptuală, se poate construi o diagramă ER care să includă entitățile de mai sus și relațiile între ele. Diagramele ER sunt esențiale pentru vizualizarea modului în care entitățile (tabelele) comunică și sunt legate prin atribute comune [4]. Exemplul următor prezintă un model generic de sistem de rezervare feroviară (în limba engleză), ilustrând câteva entități și relațiile dintre ele:

[4] *Fig.1: Exemplu de diagramă ER pentru sistemul de rezervări feroviare (entități: PAX_info (călător), Ticket_reservation, Train, Station, Seat_availability, Class, Zone etc.)* [4].

În proiectul nostru, diagrama ER ar conține entități precum **Calator**, **Rezervare**, **Bilet**, **Tren**, **Operator**, **Traseu**, **Statie**, **TipTren**, **CategorieTarif**, **Tarif** etc., și legăturile definite mai sus, cu cardinalitățile corespunzătoare.

7. Diagrama conceptuală derivată (relațională)

La nivel relațional, entitățile devin tabele, iar relațiile M:N sunt implementate prin tabele asociative (ex. *Statie_Traseu*, *Operator_Traseu*). Diagrama conceptuală derivată include toate tabelele finalizate, cu cheile primare (PK) și străine (FK) indicate. Spre exemplu, fiecare entitate de mai sus devine o tabelă cu atributele sale, iar liniile între ele reprezintă FK-PK. Un număr total de cel puțin 12 tabele (entități și asocieri) este necesar, astfel încât modelul relațional este complet.

8. Scheme relaționale

În continuare enumerăm schemele relaționale (numele tabelelor și atributele lor principale, PK și FK). Notăm între paranteze cheile primare (PK) și folosim săgeți pentru referințe (FK).

- **Operator**(*IDOperator*, NumeOperator, Tip) PK(IDOperator).
- **Traseu**(*IDTraseu*, NumeTraseu) PK(IDTraseu).
- **Statie**(*IDStatie*, NumeStatie, Oras) PK(IDStatie).
- **TipTren**(*IDTipTren*, Descriere) PK(IDTipTren).
- **Tren**(*IDTren*, NumeTren, Capacitate, IDTipTren→TipTren, IDOperator→Operator) PK(IDTren).
- **Calator**(*IDCalator*, Nume, Prenume, DataNasterii) PK(IDCalator).
- **Rezervare**(*IDRezervare*, IDCalator→Calator, DataRezervare) PK(IDRezervare).
- **CategorieTarif**(*IDCategorie*, Descriere, ProcentReducere) PK(IDCategorie).
- **Tarif**(*IDTarif*, IDTraseu→Traseu, IDCategorie→CategorieTarif, Pret) PK(IDTarif).
- **Bilet**(*IDBilet*, IDRezervare→Rezervare, IDTren→Tren, IDTraseu→Traseu, IDStatiePlecare→Statie, IDStatieSosire→Statie, IDTarif→Tarif, DataEmitere, PretBilet) PK(IDBilet).
- **Operator_Traseu**(*IDOperator→Operator*, *IDTraseu→Traseu*) PK(IDOperator, IDTraseu). (tabel asociativ)
- **Statie_Traseu**(*IDTraseu→Traseu*, *IDStatie→Statie*, Ordine) PK(IDTraseu, IDStatie). (tabel asociativ pentru succesiunea stațiilor)

În schemele relaționale de mai sus, săgețile indică FK și PK sunt subliniate sau specificate după „PK”.

9. Normalizare până la 3NF

Procesul de normalizare asigură eliminarea redundanței și a dependențelor funcționale nedorite. Conform teoriei, fiecare tabel trebuie cel puțin în 1NF (coloane atomice), 2NF (fără dependențe parțiale) și 3NF (fără dependențe tranzitive) [5] [6].

- **1NF (Prima formă normală)**: Fiecare câmp trebuie să conțină o valoare atomică. De exemplu, într-o versiune *ne-normalizată*, am fi putut stoca într-un singur tabel atât date despre rezervare cât și despre călător (împachetând nume de călător în același rând). În 1NF ne asigurăm că nu sunt grupuri de câmpuri repetate. Concret, extragem informațiile despre călător într-un tabel separat (*Calator*), legându-l prin *IDCalator*. (Prin aceasta, fiecare coloana conține câte o valoare atomică.) [5]
- **2NF**: Se elimină dependențele parțiale. Tabelele cu chei compuse nu au în acest caz atribute ce depind doar de o parte a cheii. În modelul actual, foarte puține chei sunt compuse, deci majoritatea tabelelor erau deja în 2NF.
- **3NF**: Se elimină dependențele tranzitive. Astfel, în tabelul rezultat fiecare atribut ne-cheie depinde numai de cheia primară. De exemplu, presupunem (ipotetic) că în *Rezervare* avem coloane

NumeCalator și *PrenumeCalator* care depind de călător. Aceasta ar fi o dependență tranzitivă (NumeCalator depinde de *IDCalator*, care este cheia). Normalizăm prin mutarea acestor coloane în tabelul *Calator*. Atunci în *Rezervare* rămâne doar *IDCalator* ca referință, și datele despre călător sunt stocate o singură dată în tabelul *Calator*. Prin aceasta, obținem 3NF: toate atributele non-cheie depind direct doar de cheia tabelului [6].

Exemplu de transformare:
- În 1NF, am fi avut un tabel îmbinat, de ex. `Rezervare_ne` (*IDRez*, *DataRez*, *IDCalator*, *NumeCalator*, *PrenumeCalator*). Coloanele *NumeCalator*, *PrenumeCalator* sunt informații redundante (depind de călător), încălcând 3NF.
- În 2NF, dacă `IDRez` ar fi formată din două coloane, am elimina dependențele parțiale; însă aici nu sunt atribute care să depindă doar de o parte a unei chei compuse.
- În 3NF, extragem *NumeCalator*, *PrenumeCalator* din `Rezervare_ne` într-un tabel *Calator*, astfel încât `Rezervare(IDRez, DataRez, IDCalator)` conține doar FK către *Calator*. Astfel datele călătorului nu se mai repetă la fiecare rezervare [6], iar schema finală e în 3NF.

10. Secvență pentru inserarea de date

În Oracle, putem defini o secvență pentru generarea automată a identificatorilor primari. De exemplu:

```
CREATE SEQUENCE seq_id START WITH 1 INCREMENT BY 1;
```

Această secvență (`seq_id`) poate fi folosită la inserarea datelor astfel: `VALUES (seq_id.NEXTVAL, ...)` pentru coloanele PK. Secvențele asigură generarea unui nou `ID` unic la fiecare inserare, facilitând popularea tabelelor.

11. Crearea tabelelor și inserarea datelor

Mai jos sunt exemple de comenzi SQL pentru crearea schemelor și inserarea unor date minimale în fiecare tabel. Se vor insera cel puțin 5 înregistrări în tabelele *nesociative* și cel puțin 10 în tabelele *asociative*. (În exemplele de mai jos numele și datele sunt fictive.)

```
-- Creare tabele
CREATE TABLE Operator(
  IDOperator   INT PRIMARY KEY,
  NumeOperator VARCHAR2(50) NOT NULL,
  Tip          VARCHAR2(10) DEFAULT 'Public' CHECK (Tip IN ('Public','Privat'))
);

CREATE TABLE Traseu(
  IDTraseu   INT PRIMARY KEY,
  NumeTraseu VARCHAR2(50) NOT NULL
);

CREATE TABLE Statie(
```

```
  IDStatie   INT PRIMARY KEY,
  NumeStatie VARCHAR2(50) NOT NULL,
  Oras       VARCHAR2(50) NOT NULL
);

CREATE TABLE TipTren(
  IDTipTren INT PRIMARY KEY,
  Descriere VARCHAR2(20) NOT NULL
);

CREATE TABLE Tren(
  IDTren      INT PRIMARY KEY,
  NumeTren    VARCHAR2(50),
  Capacitate  NUMBER,
  IDTipTren   INT NOT NULL,
  IDOperator  INT NOT NULL,
  FOREIGN KEY(IDTipTren) REFERENCES TipTren(IDTipTren),
  FOREIGN KEY(IDOperator) REFERENCES Operator(IDOperator)
);

CREATE TABLE Calator(
  IDCalator   INT PRIMARY KEY,
  Nume        VARCHAR2(50) NOT NULL,
  Prenume     VARCHAR2(50) NOT NULL,
  DataNasterii DATE
);

CREATE TABLE Rezervare(
  IDRezervare   INT PRIMARY KEY,
  IDCalator     INT NOT NULL,
  DataRezervare DATE DEFAULT SYSDATE,
  FOREIGN KEY(IDCalator) REFERENCES Calator(IDCalator)
);

CREATE TABLE CategorieTarif(
  IDCategorie     INT PRIMARY KEY,
  Descriere       VARCHAR2(20) NOT NULL,
  ProcentReducere NUMBER DEFAULT 0
);

CREATE TABLE Tarif(
  IDTarif     INT PRIMARY KEY,
  IDTraseu    INT NOT NULL,
  IDCategorie INT NOT NULL,
  Pret        NUMBER NOT NULL,
  FOREIGN KEY(IDTraseu) REFERENCES Traseu(IDTraseu),
  FOREIGN KEY(IDCategorie) REFERENCES CategorieTarif(IDCategorie)
);
```

```sql
CREATE TABLE Bilet(
  IDBilet        INT PRIMARY KEY,
  IDRezervare    INT,
  IDTren         INT NOT NULL,
  IDTraseu       INT NOT NULL,
  IDStatiePlecare INT NOT NULL,
  IDStatieSosire  INT NOT NULL,
  IDTarif        INT NOT NULL,
  DataEmitere    DATE DEFAULT SYSDATE,
  PretBilet      NUMBER,
  FOREIGN KEY(IDRezervare) REFERENCES Rezervare(IDRezervare),
  FOREIGN KEY(IDTren) REFERENCES Tren(IDTren),
  FOREIGN KEY(IDTraseu) REFERENCES Traseu(IDTraseu),
  FOREIGN KEY(IDStatiePlecare) REFERENCES Statie(IDStatie),
  FOREIGN KEY(IDStatieSosire) REFERENCES Statie(IDStatie),
  FOREIGN KEY(IDTarif) REFERENCES Tarif(IDTarif)
);

-- Tabele asociative
CREATE TABLE Operator_Traseu(
  IDOperator INT NOT NULL,
  IDTraseu   INT NOT NULL,
  PRIMARY KEY(IDOperator, IDTraseu),
  FOREIGN KEY(IDOperator) REFERENCES Operator(IDOperator),
  FOREIGN KEY(IDTraseu) REFERENCES Traseu(IDTraseu)
);

CREATE TABLE Statie_Traseu(
  IDTraseu INT NOT NULL,
  IDStatie INT NOT NULL,
  Ordine   NUMBER NOT NULL,
  PRIMARY KEY(IDTraseu, IDStatie),
  FOREIGN KEY(IDTraseu) REFERENCES Traseu(IDTraseu),
  FOREIGN KEY(IDStatie) REFERENCES Statie(IDStatie)
);

-- Inserări exemple (minim 5 în fiecare tabel nesociativ)
INSERT INTO Operator VALUES (1, 'CFR Calatori', 'Public');
INSERT INTO Operator VALUES (2, 'RegioTrans', 'Privat');
INSERT INTO Operator VALUES (3, 'InterRail', 'Privat');
INSERT INTO Operator VALUES (4, 'National Trains', 'Public');
INSERT INTO Operator VALUES (5, 'EuroRail', 'Privat');

INSERT INTO Traseu VALUES (10, 'Bucuresti - Brasov');
INSERT INTO Traseu VALUES (11, 'Cluj - Timisoara');
INSERT INTO Traseu VALUES (12, 'Iasi - Bucuresti');
INSERT INTO Traseu VALUES (13, 'Timisoara - Arad');
```

```sql
INSERT INTO Traseu VALUES (14, 'Brasov - Constanta');

INSERT INTO Statie VALUES (100, 'Bucuresti Nord', 'Bucuresti');
INSERT INTO Statie VALUES (101, 'Brasov', 'Brasov');
INSERT INTO Statie VALUES (102, 'Cluj Napoca', 'Cluj');
INSERT INTO Statie VALUES (103, 'Timisoara Nord', 'Timisoara');
INSERT INTO Statie VALUES (104, 'Iasi Nord', 'Iasi');

INSERT INTO TipTren VALUES (20, 'Regio');
INSERT INTO TipTren VALUES (21, 'InterRegio');
INSERT INTO TipTren VALUES (22, 'Accelerat');
INSERT INTO TipTren VALUES (23, 'Personal');
INSERT INTO TipTren VALUES (24, 'Rapid');

INSERT INTO Tren VALUES (1000, 'IC 123', 200, 21, 1);
INSERT INTO Tren VALUES (1001, 'R 45', 150, 22, 2);
INSERT INTO Tren VALUES (1002, 'P 10', 100, 20, 1);
INSERT INTO Tren VALUES (1003, 'IC 200', 300, 21, 4);
INSERT INTO Tren VALUES (1004, 'IR 67', 250, 23, 3);

INSERT INTO Calator VALUES (500, 'Ionescu', 'Andrei', DATE '1985-07-12');
INSERT INTO Calator VALUES (501, 'Popescu', 'Maria', DATE '1990-03-05');
INSERT INTO Calator VALUES (502, 'Vasilescu', 'Ioana', DATE '2000-12-20');
INSERT INTO Calator VALUES (503, 'Georgescu', 'Mihai', DATE '1975-11-30');
INSERT INTO Calator VALUES (504, 'Dobre', 'Elena', DATE '1968-01-15');

INSERT INTO Rezervare VALUES (600, 500, DATE '2025-05-01');
INSERT INTO Rezervare VALUES (601, 501, DATE '2025-04-20');
INSERT INTO Rezervare VALUES (602, 502, DATE '2025-04-22');
INSERT INTO Rezervare VALUES (603, 503, DATE '2025-05-02');
INSERT INTO Rezervare VALUES (604, 500, DATE '2025-04-15');

INSERT INTO CategorieTarif VALUES (30, 'Adult', 0);
INSERT INTO CategorieTarif VALUES (31, 'Student', 25);
INSERT INTO CategorieTarif VALUES (32, 'Copil', 50);
INSERT INTO CategorieTarif VALUES (33, 'Pensionar', 15);
INSERT INTO CategorieTarif VALUES (34, 'Elev', 50);

INSERT INTO Tarif VALUES (700, 10, 30, 100.0);
INSERT INTO Tarif VALUES (701, 10, 31, 75.0);
INSERT INTO Tarif VALUES (702, 11, 30, 80.0);
INSERT INTO Tarif VALUES (703, 11, 31, 60.0);
INSERT INTO Tarif VALUES (704, 12, 30, 120.0);

-- Tabele asociative (minim 10 inserări fiecare)
INSERT INTO Operator_Traseu VALUES (1, 10);
INSERT INTO Operator_Traseu VALUES (2, 10);
INSERT INTO Operator_Traseu VALUES (1, 11);
```

```
INSERT INTO Operator_Traseu VALUES (3, 11);
INSERT INTO Operator_Traseu VALUES (4, 12);
INSERT INTO Operator_Traseu VALUES (5, 12);
INSERT INTO Operator_Traseu VALUES (1, 12);
INSERT INTO Operator_Traseu VALUES (2, 13);
INSERT INTO Operator_Traseu VALUES (3, 14);
INSERT INTO Operator_Traseu VALUES (4, 14);

INSERT INTO Statie_Traseu VALUES (10, 100, 1);
INSERT INTO Statie_Traseu VALUES (10, 101, 2);
INSERT INTO Statie_Traseu VALUES (11, 102, 1);
INSERT INTO Statie_Traseu VALUES (11, 103, 2);
INSERT INTO Statie_Traseu VALUES (12, 104, 1);
INSERT INTO Statie_Traseu VALUES (12, 100, 2);
INSERT INTO Statie_Traseu VALUES (13, 103, 1);
INSERT INTO Statie_Traseu VALUES (13, 102, 2);
INSERT INTO Statie_Traseu VALUES (14, 101, 1);
INSERT INTO Statie_Traseu VALUES (14, 100, 2);
```

În acest fel am creat tabelele și am introdus date exemplu (maxim 30 înregistrări în fiecare, conform cerințelor). Datele permit executarea interogărilor complexe din pasul următor.

12. Interogări SQL complexe

Vom prezenta 5 cereri SQL care includ condițiile cerute:

a) Subcereri corelate (EXISTS/IN cu legături între 3 tabele): Afișează numele călătorilor care au rezervări (tabelele *Rezervare* şi *Bilet*) pe traseul cu `IDTraseu = 10`. Aceasta folosește un subselect corelat pe *Calator*, *Rezervare* și *Bilet*:

```
SELECT C.Nume, C.Prenume
FROM Calator C
WHERE EXISTS (
  SELECT 1
  FROM Rezervare R JOIN Bilet B ON R.IDRezervare = B.IDRezervare
  WHERE R.IDCalator = C.IDCalator AND B.IDTraseu = 10
);
```

În această interogare, subinterogarea dependentă *(EXISTS)* verifică pentru fiecare călător dacă există rezervări care să corespundă unui bilet pe traseul 10. Se utilizează 3 tabele (Calator, Rezervare, Bilet) într-o subcerere sincronizată.

b) Subcereri nesincronizate în FROM: Exemplu cu subquery independent în clauza FROM. Afișăm traseele și prețul mediu al tarifelor pentru fiecare traseu, folosind un subselect în FROM:

```sql
SELECT T.IDTraseu, T.NumeTraseu, A.avgPret
FROM Traseu T
JOIN (
  SELECT IDTraseu, AVG(Pret) AS avgPret
  FROM Tarif
  GROUP BY IDTraseu
) A ON T.IDTraseu = A.IDTraseu
WHERE A.avgPret > 80;
```

Aici, subinterogarea din FROM *(SELECT ... FROM Tarif GROUP BY IDTraseu)* calculează prețul mediu pe traseu (tabelele *Tarif*), iar rezultatul este apoi legat la *Traseu*. Este un subquery nesincronizat (independent) în secțiunea FROM.

c) Agregare, GROUP BY, HAVING cu subcereri: Exemplu cu agregări și subquery. Afișăm pentru fiecare operator numărul de bilete vândute (tabelele *Operator*, *Tren*, *Bilet*) și filtrăm cu HAVING utilizând un subquery nesincronizat în filtrare:

```sql
SELECT O.NumeOperator, COUNT(B.IDBilet) AS TotalBilete
FROM Operator O
JOIN Tren T ON O.IDOperator = T.IDOperator
JOIN Bilet B ON T.IDTren = B.IDTren
GROUP BY O.NumeOperator
HAVING COUNT(B.IDBilet) > (
  SELECT AVG(cnt)
  FROM (
    SELECT COUNT(*) AS cnt
    FROM Operator O2
    JOIN Tren T2 ON O2.IDOperator = T2.IDOperator
    JOIN Bilet B2 ON T2.IDTren = B2.IDTren
    GROUP BY O2.IDOperator
  )
);
```

Acestă interogare grupează după operator și numără biletele. Condiția HAVING compară acest număr cu valoarea medie calculată de subquery-ul intern. Subquery-urile nesincronizate se regăsesc în clauza HAVING și din nou implică 3 tabele. Funcțiile agregate (`COUNT` , `AVG`) și filtrul HAVING răspund cerinței.

d) ORDONARE + NVL + DECODE: Exemplu cu DECODE și NVL în SELECT. Afișăm id-ul biletului, categoria tarifului descriptivă (folosind DECODE pe tabelele *Bilet* și *CategorieTarif*), și prețul (folosind NVL pentru a înlocui NULL cu 0), apoi ordonăm descrescător după preț:

```sql
SELECT B.IDBilet,
       DECODE(C.Descriere, 'Adult','Tarif Adult', 'Student','Tarif Student',
'Copil','Tarif Copil', 'Alt') AS CategorieDescriere,
```

```
       NVL(B.PretBilet, 0) AS PretBilet
FROM Bilet B
JOIN Tarif T ON B.IDTarif = T.IDTarif
JOIN CategorieTarif C ON T.IDCategorie = C.IDCategorie
ORDER BY NVL(B.PretBilet, 0) DESC;
```

În acest exemplu se realizează *JOIN* între Bilet, Tarif și CategorieTarif (3 tabele). Funcția `DECODE` traduce categoria de tarif într-un text, iar `NVL` înlocuiește prețul NULL cu 0. Rezultatele sunt sortate (ORDER BY) după preț descrescător.

e) Funcții pe șir și date + CASE: Exemplu de interogare care combină două funcții de șir, două funcții de dată și o expresie CASE. Afișăm pentru fiecare călător: numele complet (concatenare), primele 3 litere ale prenumelui, data ultimei rezervări formatată, diferența de zile față de azi, și vârsta aproximativă (folosind CASE). Tabele: *Calator* și *Rezervare*:

```
SELECT C.IDCalator,
       C.Nume || ' ' || C.Prenume AS NumeComplet,
       SUBSTR(C.Prenume,1,3) AS InitPrenume,
       TO_CHAR(MAX(R.DataRezervare), 'YYYY-MM-DD') AS UltimaRezervare,
       TRUNC(SYSDATE - MAX(R.DataRezervare)) AS ZilePanaAzi,
       CASE
         WHEN C.DataNasterii IS NOT NULL
         THEN EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM C.DataNasterii)
         ELSE NULL
       END AS VarstaAproximativa
FROM Calator C
JOIN Rezervare R ON C.IDCalator = R.IDCalator
GROUP BY C.IDCalator, C.Nume, C.Prenume, C.DataNasterii;
```

Aici folosim: două funcții pe șir (`||` pentru concatenare, `SUBSTR`), două funcții pe dată (`TO_CHAR`, calcul diferență de zile cu `SYSDATE`), și o expresie `CASE` pentru calculul vârstei. Interogarea grupează rezultatele după călător.

f) Bloc WITH (CTE) și o altă cerință: Exemplu cu CTE (Common Table Expression). Calculăm data ultimei rezervări per călător, apoi selectăm toți călătorii cu rezervări recente:

```
WITH RezUltime AS (
  SELECT IDCalator, MAX(DataRezervare) AS DataUltima
  FROM Rezervare
  GROUP BY IDCalator
)
SELECT C.IDCalator, C.Nume, C.Prenume, R.DataUltima
FROM RezUltime R
```

```
JOIN Calator C ON C.IDCalator = R.IDCalator
WHERE R.DataUltima >= ADD_MONTHS(SYSDATE, -1);
```

CTE-ul `RezUltime` conține pentru fiecare călător data ultimei rezervări. Apoi selectăm călătorii care au făcut rezervări în ultimele 30 de zile. Acest exemplu include un bloc `WITH` și demonstrează utilizarea expresiei `CASE` sau a altor funcții (împlinită în interogarea de mai sus).

13. Operații de actualizare și ștergere cu subcereri

Câteva exemple de operații UPDATE/DELETE care folosesc subinterogări:

- **Ștergere (DELETE)**: Eliminăm biletele care nu aparțin nicioia dintre rezervările existente (integritate referențială):

  ```
  DELETE FROM Bilet
  WHERE IDRezervare NOT IN (SELECT IDRezervare FROM Rezervare);
  ```

- **Actualizare (UPDATE)**: Marcăm ca șters toți călătorii care nu au nici o rezervare asociată (folosind un subquery):

  ```
  UPDATE Calator
  SET Nume = Nume || ' (inactiv)'
  WHERE IDCalator NOT IN (SELECT IDCalator FROM Rezervare);
  ```

- **UPDATE cu condiție complexă**: Majorăm tarifele traseului 10 cu 10% pentru categoria de student:

  ```
  UPDATE Tarif
  SET Pret = Pret * 1.10
  WHERE IDTraseu = 10 AND IDCategorie = 31;
  ```

  (aici condiția ar putea include un subquery, de exemplu `WHERE IDTraseu IN (SELECT IDTraseu FROM Operator_Traseu WHERE IDOperator=2)` *).*

Toate exemplele de mai sus folosesc subinterogări fie în clauza WHERE (subselect IN/NOT IN) sau pot folosi EXISTS/NOT EXISTS pentru a alege înregistrările în funcție de date din alte tabele.

14. Vedere (VIEW) complexă + DML permis/ nepermis

Creăm un view complex care unește mai multe tabele:

```
CREATE VIEW V_BileteOperator AS
SELECT O.IDOperator, O.NumeOperator, R.IDRezervare, B.IDBilet, B.PretBilet
FROM Operator O
```

```
JOIN Tren T ON O.IDOperator = T.IDOperator
JOIN Bilet B ON T.IDTren = B.IDTren
LEFT JOIN Rezervare R ON B.IDRezervare = R.IDRezervare;
```

Acest *view* include coloane din tabelele *Operator*, *Tren*, *Bilet* și *Rezervare*.

- **DML permis**: În general, actualizările (`UPDATE`, `DELETE`) sunt permise pe coloane provenite dintr-un singur tabel de bază (care prezintă cheia primară) [7]. De exemplu, se poate face:

  ```
  UPDATE V_BileteOperator
  SET PretBilet = PretBilet * 1.05
  WHERE IDOperator = 1;
  ```

  dacă `IDOperator=1` corespunde unui singur operator, doar biletele din acel operator se actualizează.
- **DML nepermis**: Inserțiile nu sunt permise pe un view care nu conține toate coloanele tabelelor de bază sau care include multe tabele. În exemplul de mai sus, `V_BileteOperator` nu permite `INSERT` direct deoarece nu sunt furnizate toate coloanele cheie ale tabelelor subiacente. De asemenea, view-ul cu `JOIN` este probabil *ne-updatabil*, deci comenzi precum `INSERT INTO V_BileteOperator(...) VALUES(...)` vor fi respinse [7].

Astfel, actualizarea prin view este permisă doar dacă nu afectează mai multe tabele simultan și respectă regulile de view updatability. Conform documentației, un view cu mai multe tabele (join) **nu** permite de regulă operații de inserare sau ștergere [7], iar UPDATE-ul se aplică de obicei pe tabelul principal care îndeplinește condițiile.

15. Cereri separate specializate

- Outer join pe ≥4 tabele: Un exemplu care leagă 5 tabele:

```
SELECT C.Nume, C.Prenume, R.IDRezervare, B.IDBilet, T.NumeTraseu, O.NumeOperator
FROM Calator C
LEFT JOIN Rezervare R ON C.IDCalator = R.IDCalator
LEFT JOIN Bilet B ON R.IDRezervare = B.IDRezervare
LEFT JOIN Tren T ON B.IDTren = T.IDTren
LEFT JOIN Operator O ON T.IDOperator = O.IDOperator;
```

Această interogare utilizează *LEFT OUTER JOIN* pe 5 tabele (Calator, Rezervare, Bilet, Tren, Operator) și afișează informații combinate. Pentru oricare călător fără rezervări, coloanele din tabelele alăturate vor fi NULL.

- Operația de diviziune (÷): Exemplu „revenit” pentru SQL: Afișează toți călătorii care au cumpărat bilete pe **toate** traseele din lista (10,11,12). Putem emula diviziunea cu două subinterogări `NOT EXISTS`:

```sql
SELECT C.IDCalator, C.Nume, C.Prenume
FROM Calator C
WHERE NOT EXISTS (
  SELECT IDTraseu
  FROM Traseu T2
  WHERE T2.IDTraseu IN (10, 11, 12)
    AND NOT EXISTS (
       SELECT 1
       FROM Rezervare R2 JOIN Bilet B2 ON R2.IDRezervare = B2.IDRezervare
       WHERE R2.IDCalator = C.IDCalator AND B2.IDTraseu = T2.IDTraseu
    )
);
```

Această interogare întoarce călătorii pentru care nu există niciun traseu (din cele 10,11,12) la care să nu fi cumpărat bilet.

- Top-N Analysis: Exemplu de interogare *Top-N* (în Oracle, folosind ROWNUM): găsim primii 3 operatori după numărul de bilete vândute:

```sql
SELECT Operator_ID, TotalBilete
FROM (
  SELECT O.IDOperator AS Operator_ID, COUNT(B.IDBilet) AS TotalBilete
  FROM Operator O
  JOIN Tren T ON O.IDOperator = T.IDOperator
  JOIN Bilet B ON T.IDTren = B.IDTren
  GROUP BY O.IDOperator
  ORDER BY COUNT(B.IDBilet) DESC
)
WHERE ROWNUM <= 3;
```

Secvența interioară ordonează operatorii după numărul de bilete descrescător, iar clauza externă `WHERE ROWNUM <= 3` selectează top 3 rezultate.

16. Optimizarea unei cereri

Vom optimiza o interogare prin reordonarea operațiilor (push-down de selecții). Presupunem următoarea interogare inițială:

```sql
-- Interogare inițială (less optimal)
SELECT C.Nume, C.Prenume
FROM Calator C, Rezervare R
WHERE C.IDCalator = R.IDCalator
  AND R.DataRezervare > SYSDATE-7;
```

Expresia algebrică inițială:
- Selecție la final: σ(R.DataRezervare > SYSDATE-7) (Calator ⋈ Rezervare).

Arbore algebric inițial:

```
      π Nume, Prenume
           |
    σ(DataRezervare > SYSDATE-7)
           |
      Join (C.ID = R.ID)
      /            \
   Calator       Rezervare
```

Pentru optimizare, aplicăm *predicate pushdown*: aducem condiția `R.DataRezervare > SYSDATE-7` **înainte** de efectuarea join-ului, astfel încât să filtrăm înregistrările din *Rezervare* înainte de a le uni cu *Calator*.

Expresie algebrică optimizată:
σ(R.DataRezervare > SYSDATE-7)(Rezervare) ⋈ Calator

Arbore algebric optimizat:

```
      π Nume, Prenume
           |
        Join on C.ID=R.ID
       /          \
   Calator   σ(DataRezervare > SYSDATE-7)
               |
           Rezervare
```

SQL optimizat: Mutăm condiția în WHERE și folosim sintaxa explicită JOIN:

```sql
SELECT C.Nume, C.Prenume
FROM Rezervare R
JOIN Calator C ON C.IDCalator = R.IDCalator
WHERE R.DataRezervare > SYSDATE-7;
```

În această variantă, filtrul `R.DataRezervare > SYSDATE-7` se aplică imediat pe tabela *Rezervare*, reducând setul de date înainte de join. Conform literaturii, arborele de query și reordonarea pot accelera executarea [8] . Diagrama algebrică nouă arată că filtrarea este realizată la nodul foiței *Rezervare*, iar apoi se face join cu *Calator*. Rezultatul este o interogare echivalentă dar mai eficientă, deoarece se evaluează mai puține rânduri la join (predicate pushdown).

17. Normalizare avansată și denormalizare

a) De la 3NF la BCNF, 4NF, 5NF:
- **BCNF (Boyce-Codd Normal Form):** Formă mai strictă decât 3NF. Un tabel este în BCNF dacă pentru orice dependență funcțională X→Y, X este super-cheie. BCNF elimină situațiile de redundanță rămasă în 3NF [9]. În exemplul nostru, majoritatea tabelelor sunt deja BCNF, deoarece fiecare dependență funcțională majoră se bazează pe cheia primară. Dacă ar exista o dependență funcțională ne-cheie care nu implică cheia primară, ar trebui descompusă. (De exemplu, dacă în *Bilet* am avea două coloane care se influențează reciproc fără a fi chei, ar fi nevoie de normalizare suplimentară.)
- **4NF:** Se ocupă de dependențele *multivaluate*. Un tabel este în 4NF dacă nu conține dependențe multivaluate ne-triviale care nu implică cheia primară [10]. Practic, 4NF elimină redundanța când același rând ar putea combina valori de două ori. În schema noastră simplificată nu avem exemple evidente de MVD, dar dacă, ipotetic, un tabel ar stoca două atribute independente multilista (ex. preferințe de genuri muzicale ale unui călător), acestea ar fi mutate în tabele separate pentru 4NF [10].
- **5NF (PJ/NF):** Asigură eliminarea dependențelor de tip *join*. Un tabel este în 5NF dacă fiecare dependență de tip join este îndeplinită de cheile candidat [11]. 5NF se ocupă de redundanțe complexe care nu aparțin doar unor dependențe funcționale sau multivaluate simple. În practică, ajungem la 5NF doar dacă căutăm descompuneri maxime ale tabelelor astfel încât doar cheile pot recombina toată informația. Conform literaturii, 5NF este forma finală care asigură că nu mai există redundanțe ce pot fi eliminate prin divizarea tabelului [11].

b) Denormalizare și justificare:
În anumite scenarii operaționale, se aplică *denormalizarea* pentru a îmbunătăți performanța citirii. Aceasta presupune combinarea unor tabele separate sau adăugarea unor coloane redundante, acceptând o anumită redundanță pentru a evita join-urile costisitoare. Deși normalizarea reduce redundanța și anomaliile, în practică denormalizarea este folosită când există foarte multe interogări complexe cu join. Astfel, denormalizarea poate accelera citirile (interogările *read-heavy*) prin reducerea numărului de join-uri necesare [12].

Cu toate acestea, există un **trade-off**: datele duplicat cresc riscul de inconsistență și dificultatea întreținerii. Dezavantajul denormalizării este tocmai accentuarea redundantței și a riscului de erori la actualizare [13] [12]. Datele introduse în mai multe tabele trebuie sincronizate manual. În concluzie, se recomandă denormalizarea **când** performanța citirilor este critică și problemele de integritate pot fi gestionate (de ex. prin aplicarea controalelor la nivel de aplicație). În contextul bazei de bilete, s-ar putea denormaliza spre exemplu prin păstrarea denumirii stației în tabelul *Bilet* (pentru rapoarte rapide), dacă join-urile cu tabelul *Statie* devin un blocaj de performanță. Folosind denormalizarea cu moderație se câștigă viteză de interogare, însă conform surselor pierdem din beneficiile integrității oferite de normalizare [12] [13].

Surse bibliografice: Proiectarea modelelor entitate-relație și normalizarea au la bază lucrări de specialitate (Ileana Popescu, Leția Velcescu, *Proiectarea bazelor de date* etc.). Definițiile formelor normale și noțiunile despre cheile primare/străine le-am susținut prin referințe academice și documentație standard (Wiki, Oracle, GeeksforGeeks) pentru acuratețe [1] [6] [2] [9] [10] [11] [7] [8] [12].

---

[1] [5] Database normalization - Wikipedia
https://en.wikipedia.org/wiki/Database_normalization

2 Difference between Primary key and Unique key | GeeksforGeeks
https://www.geeksforgeeks.org/difference-between-primary-key-and-unique-key/

3 Entity Relationship Mapping
https://docs.oracle.com/cd/B10002_01/generic.903/a97677/ormap.htm

4 Online Railway Ticket Reservation System | GeeksforGeeks
https://www.geeksforgeeks.org/online-railway-ticket-reservation-system/

6 Third normal form - Wikipedia
https://en.wikipedia.org/wiki/Third_normal_form

7 27.5.3 Updatable and Insertable Views
https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/view-updatability.html

8 Query Tree in Relational Algebra | GeeksforGeeks
https://www.geeksforgeeks.org/query-tree-in-relational-algebra/

9 Boyce–Codd normal form - Wikipedia
https://en.wikipedia.org/wiki/Boyce%E2%80%93Codd_normal_form

10 Fourth normal form - Wikipedia
https://en.wikipedia.org/wiki/Fourth_normal_form

11 Fifth normal form - Wikipedia
https://en.wikipedia.org/wiki/Fifth_normal_form

12 Data Denormalization: The Complete Guide | Splunk
https://www.splunk.com/en_us/blog/learn/data-denormalization.html

13 The Trade-offs Between Database Normalization and Denormalization - DEV Community
https://dev.to/er_dward/the-trade-offs-between-database-normalization-and-denormalization-4kdo